

Memory Management Techniques

- Uniprogramming
 - The main memory is divided into two parts, one part for OS (monitor) and one part for the program currently being executed
 - Disadvantage: one program at a time.
- Fixed Partitioning
 - main memory is divided into a number of static partitions at system generation time
 - Equal size partition:
 - any process whose size is less or equal to the partition size can be loaded into any available partition.
 - Disadvantage: a program may be too big to fit into a partition
 - Main memory use is inefficient
 - Unequal-size partition
 - Assign each process to the smallest partition within which it will fit
 - One process queue per partition
 - Advantage:
 - Minimize wasted memory within partition (internal fragmentation)
 - Disadvantage: large partition may be unused, even when some smaller processes could have been assigned to it.

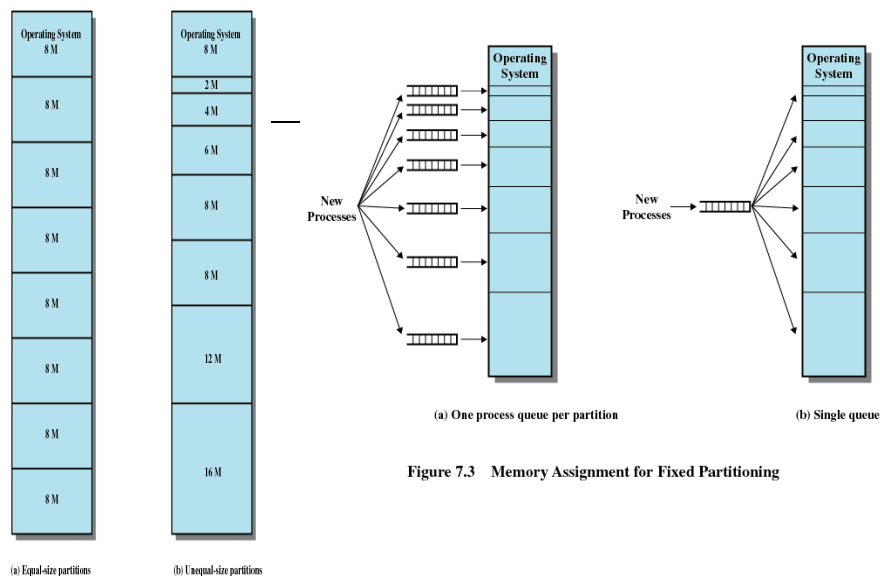
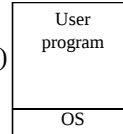


Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

Figure 7.3 Memory Assignment for Fixed Partitioning

- Single queue for all processes
 - At load time, the smallest available partition that will hold the process is selected
 - Disadvantages of fixed partitioning
 - Limit number of active processes
 - Internal fragmentation
- Dynamic partitioning (variable size)
 - The partitions used are of variable length and number. When a process is brought into main memory, it is allocated exactly as much memory as it requires and no more.
 - Advantage: use memory space efficiently
 - Disadvantage: external fragmentation, the memory that is external to all partitions are a lot of small holes in memory.
 - External fragmentation solution
 - Compaction: from time to time, OS shifts the processes so that they are contiguous and so that all the free memory is together in one block

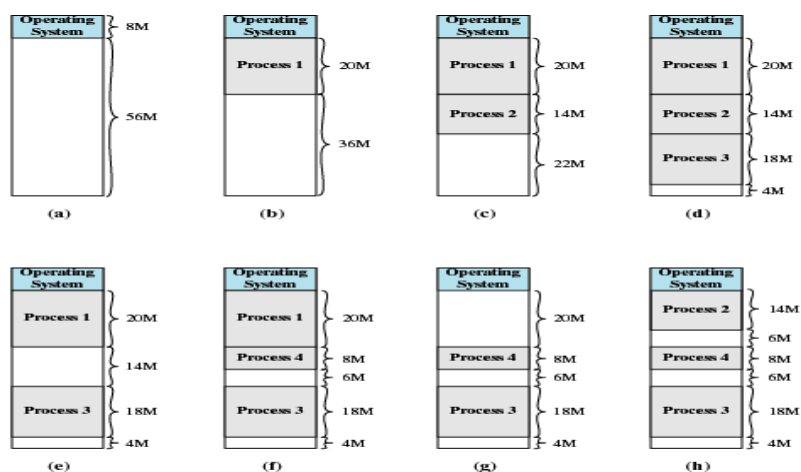
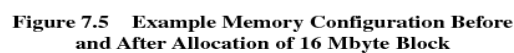


Figure 7.4 The Effect of Dynamic Partitioning

- Operating system must decide which free block to allocate to a process
- **Best-fit algorithm**
 - Chooses the block that is closest in size to the request
 - Worst performer overall
 - Since smallest block is found for process, the smallest amount of fragmentation is left
 - Memory compaction must be done more often
- **First-fit algorithm**
 - Scans memory from the beginning and chooses the first available block that is large enough
 - Fastest
- **Next-fit**
 - Scans memory from the location of the last placement
 - More often allocate a block of memory at the end of memory where the largest block is found
 - The largest block of memory is broken up into smaller blocks
 - Compaction is required to obtain a large block at the end of memory



Segmentation

- ❑ All segments of all programs do not have to be of the same length
- ❑ There is a maximum segment length
- ❑ Addressing consist of two parts - a segment number and an offset
- ❑ Since segments are not equal, segmentation is similar to dynamic partitioning

1.7

Paging

- ❑ Partition memory into small equal fixed-size chunks and divide each process into the same size chunks
- ❑ The chunks of a process are called pages and chunks of memory are called frames
- ❑ Operating system maintains a page table for each process
 - Contains the frame location for each page in the process
 - Memory address consist of a page number and offset within the page
- ❑ Combined paging and segmentation

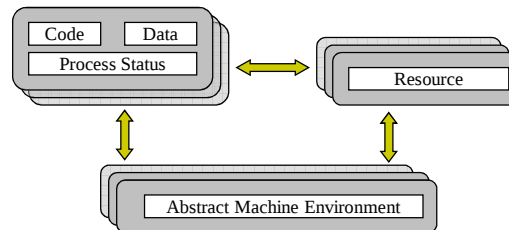
address

Segment no	Page no	Offset no
------------	---------	-----------

1.8

Threading

- A **process** is a sequential program in execution.
- A **process** is a unit of computation.
- **Process components:**
 - The program (code) to be executed.
 - The data on which the program will execute.
 - Resources required by the program.
 - The status of the process execution.
- A process runs in an abstract machine environment (could be OS) that manages the sharing and isolation of resources among the community of processes.



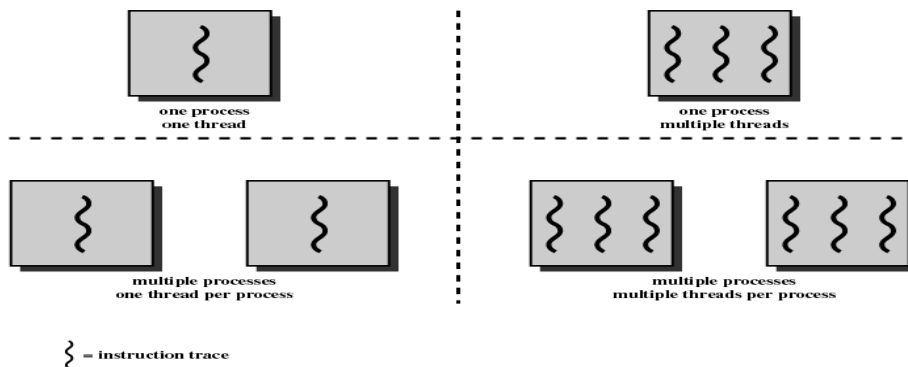
Program and Process

- A program is a static entity made up of program statements. The process defines the run-time behavior (the execution of the program)
- A process is a dynamic entity that executes a program on a particular set of data.
- Two or more processes could execute the same program, each using their own data and resources.
- Task manager

- **A thread:** is an independent execution path, able to run simultaneously with other threads.
 - **Multithreading :** allows an application to have multiple threads of execution running concurrently. (ex, word , spell checking)
- In .NET a program starts in a single thread created automatically by the CLR (Common Language Runtime) and operating system (the “main” thread), and is made multithreaded by creating additional threads
- CLR assigns each thread its own memory stack so that local variables are kept separate
- Threads share data if they have a common reference to the same data
- All threads within a single application are logically contained within a process
- The key difference between threads and processes:
 - Processes are fully isolated from each other;
 - Threads share memory with other threads running in the same application'''

Processes and Threads implementation

- MS-DOS supports a single thread
- UNIX supports multiple user processes but only supports one thread per process
- Windows, Solaris, Linux, Mach, and OS/2 support multiple threads within a single process.



Threading Models

There are typically two threading models supported by OS:

- **Cooperative Threading Model;**
- **Preemptive Threading Model.**

- **Cooperative Threading Model**

- In a cooperative system, a thread retains control of the processor until it decides to give it up (which might be never).
- Supporting OS – Windows 3.x, Solaris, Mac OS.
- The various threads have to cooperate with each other. If not, some of them will be starving (never given a chance to run –**starvation**)
- Scheduling in most cooperative systems is done strictly by priority level - when the current thread gives up control, the highest-priority waiting thread gets control.

Preemptive Threading Model

- In a preemptive system, some sort of timer is used by the operating system itself to cause a context swap.
- Supporting OS – Windows ,Solaris, Linux.
- When the timer "ticks" the OS can abruptly take control away from the running thread and give control to another thread.
- The interval between timer ticks is called a time slice.
- To get to concurrency, the OS must do the thread scheduling.
- Preemptive systems are less efficient than cooperative ones because the thread management must be done by the OS' kernel, but they are easier to program (except their synchronization).

OpenMP parallel programming model

- ❑ OpenMP is a collection of compiler directives and library functions that are used to create parallel programs for shared-memory computers.
 - MP stands for \multi-processing.
- ❑ OpenMP is combined with C, C++, or Fortran to create a multithreading programming language, in which all processes are assumed to share a single address space.
- ❑ OpenMP is based on the fork / join programming model: all programs start as a single (master) thread, fork additional threads where parallelism is desired (the parallel region), then join back together.
- ❑ The threads must synchronize before joining.

110

- ❑ OpenMP library: `#include <omp.h>`
- ❑ `#pragma omp parallel`
 - {
 - /*Parallel Section*/
 - }
- ❑ `omp_get_num_threads();`
 - Return an integer value that is the number of threads created by OpenMP in the run.
- ❑ `omp_get_thread_num()`
 - Return an integer value that is a unique thread id number for each thread

111


```

#include<omp.h>
#include<iostream>
using namespace std;

int main()
{
#pragma omp parallel

cout << "\n Hello from thread " << omp_get_thread_num()<< "  out
  of  " << omp_get_num_threads()<<endl;
}

```

117

<pre> #include<omp.h> #include<iostream> using namespace std; int main() { int tid; #pragma omp parallel private(tid) { tid = omp_get_thread_num(); #pragma omp sections { #pragma omp section cout << "\n Hello from thread " << tid; #pragma omp section cout << "\n Hello from thread " << tid; #pragma omp section cout << "\n Hello from thread " << tid; } } } </pre>	<pre> #pragma omp section cout << "\n Hello from thread " << tid; #pragma omp section cout << "\n Hello from thread " << tid; #pragma omp section cout << "\n Hello from thread " << tid; #pragma omp section cout << "\n Hello from thread " << tid; #pragma omp section cout << "\n Hello from thread " << tid; #pragma omp section cout << "\n Hello from thread " << tid; #pragma omp section cout << "\n Hello from thread " << tid; } } } </pre>
---	--

118